Chapter 1 – Designing a Database

Introduction

If you are setting out to do a project using MS Access, then you need to have covered some of the theory of database design. In this chapter you will briefly review some of the terminology used and look at how to approach the design of a database once you have selected a suitable topic or problem for your project.

A database is basically a collection of data. There are many different types of database, but the two that you are most likely to come across in this course are *flat file* databases and *relational* databases.

A *flat file* database consists of one or more unrelated tables. A *table* in a database is a collection of records which can be viewed in rows and columns. Thus for example you could keep a flat file database of all members of a Club:

Club Member

Member ID	Surname	First Name	Member Type	Date Joined	Subs Paid
123	Bell	Sarah	Junior	12/01/97	Y
124	Naylor	Paul	Senior	18/01/99	Y
125	Townsend	Roxanne	Senior	16/04/98	N
etc					

Club Member is the *entity* about which data is being held.

Member ID, **Surname**, **First Name**, **Member Type**, **Date Joined** and **Subs Paid** are *attributes* of this entity.

Member ID acts as the *primary key*; it is the attribute which uniquely identifies a row of the table.

A flat file database can be quite useful in a rather limited way; for example, you can use it to search for all Junior Members, or all members who have not yet paid their annual subscription, and then merge this subset of members with a letter to remind them to pay up (a *mail-merge*).

However, a flat file database is not suitable for an Advanced Level project as it does not offer sufficient scope to use the advanced features of Access. Access is a *relational* database package, meaning that you can construct a database consisting of several tables which are related to each other by means of common fields.

Entity-relationship diagrams

There are three different possible *degrees of relationship* between two entities. These are:

One-to-one e.g. Husband and wife. A husband can have one wife, and a wife can have one husband.

One-to-many e.g. Football team and player. A team has many players, but a player belongs to only one team.

Many-to-many e.g. Product and component. A product has many components, and the same component can be used in many different products.

These relationships can be shown in an *entity-relationship (E-R) diagram* as follows:

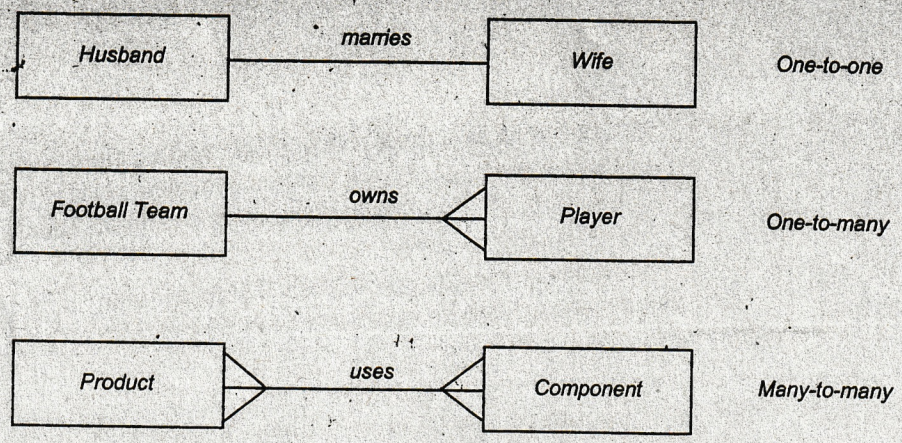

Figure 1.1: Different degrees of relationship

Attributes and key fields

Each entity will have attributes associated with it. An *attribute* is a property or characteristic, for example the entity Product will have attributes such as Product ID, Description, Cost Price. Sometimes it is hard to separate the entities and the attributes – Component, for example, is an entity in the above diagram but each component is also an attribute of a particular product.

Each entity in a database must have one or more attributes called a *primary key* which uniquely identifies it – for example Product ID would be an obvious primary key of Product.

Normalisation

Normalisation is a process used to design a database in the most efficient way, and you need to make sure you understand the process before you start designing your database. Basically, there are three stages or rules to be applied known as First, Second and Third Normal Form. Once you are familiar with these rules the whole process of normalisation seems very much like just using a bit of common sense. We'll look briefly at the three stages of normalisation.

In designing a database to hold information about students and subjects, for example, you might start by identifying the two entities STUDENT and SUBJECT. These entities are joined in a many-to-many relationship, since a student may study several subjects, and the same subject may be studied by several students. You can then draw an entity-relationship diagram as shown below:

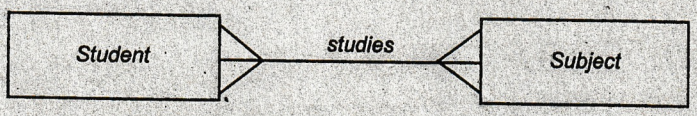

Figure 1.2: Initial stage in database design

First Normal Form

A database in *first normal form* must not contain repeating attributes.

The STUDENT entity will probably have attributes StudentNumber, Surname, FirstName, DateOfBirth, DateEnrolled, and possibly many other attributes as well, depending on the

Note: Names of attributes and other objects can be up to 64 characters long in Access, and it is permissible to have spaces or other characters in an attribute name. However in this book attribute names will be written without spaces, and this is recommended. In any case you would be well advised to decide whether or not to use spaces, and then apply your convention consistently.

objectives of the database application and the information we want out of it. The subjects that a student studies, for example, are attributes of a student.

The standard notation for representing a table is to put the entity name in uppercase letters, followed by the attributes within parentheses, with the unique primary key underlined, as shown below:

STUDENT (StudentNumber, Surname, FirstName, DateOfBirth, DateEnrolled)

What we must NOT do is to put the subjects studied by a particular student in the same table, because a student may study more than one subject. This is what is meant by a *repeating attribute*. In other words, do NOT design the table like this:

STUDENT (StudentNumber, Surname, FirstName, DateOfBirth, DateEnrolled, Subject1, Subject2, Subject3)

You can probably see the reasons for not designing a table this way. For one thing, what would happen if a student studies more than 3 subjects? For another, it would be very difficult to pick out all students doing a particular subject, because you wouldn't know which column to look in.

The STUDENT table can be left as it was originally. **Subject** is an entity and gets a table of its own. After further investigation of the problem, the designer may decide that attributes of Subject which need to be recorded include the subject code (a unique ID), the subject name, hours per week and the ID and name of the tutor in charge of that particular subject.

SUBJECT (SubjectCode, SubjectName, HrsPerWeek, TutorID, TutorName)

(but we'll be modifying this shortly, so if you've spotted a weakness, don't worry!)

Next, we need to introduce a new table to show the subjects that each student is studying.

STUDENT_SUBJECT (StudentNumber, SubjectCode)

Notice that in this table, both the StudentNumber and the SubjectCode are needed to form a unique primary key. This can be referred to as a *composite key*. If a student studies 3 subjects, there will be three records in the table specifying the 3 different subject codes for that student.

Second Normal Form

A table is in *second normal form* if it is in first normal form and no column that is not part of a primary key is dependent on only a portion of the primary key.

A bit of a mouthful, but it all boils down to common sense. What it's saying is, don't include attributes in tables that aren't needed in that table. For example, suppose we had designed the last table as

STUDENT_SUBJECT (StudentNumber, Surname, FirstName, SubjectCode, SubjectName)

Surname and FirstName are not part of the primary key, and they depend only on StudentNumber. Given a particular StudentNumber, you can look up the STUDENT table to find out the corresponding Surname and FirstName, so they are not needed in the STUDENT_SUBJECT table. The same logic applies to SubjectName.

Third Normal Form

A table is in *third normal form* if it contains no 'non-key dependencies'.

Look again at the SUBJECT table. We have specified two attributes TutorID and TutorName in this table. However, TutorID is enough to identify the subject tutor, and TutorName depends on TutorID, not on SubjectCode. Therefore it should not be in this table. We have in fact identified another entity, TUTOR.

The designer would need to find out more about tutors. Is a tutor responsible for a student or a subject? Can a tutor be responsible for more than one student or subject? Can a student or subject have more than one tutor?

We will assume that it has been established that each subject is under the administration of just one tutor who acts as subject leader, but that a single tutor may be responsible for more than one subject. For example Mrs Bell may be responsible for both Computing and Information Technology. Of course, this may not be true in some schools – the correct relationship would have to be established when the problem was being analysed.

The entity-relationship diagram may be drawn as follows:

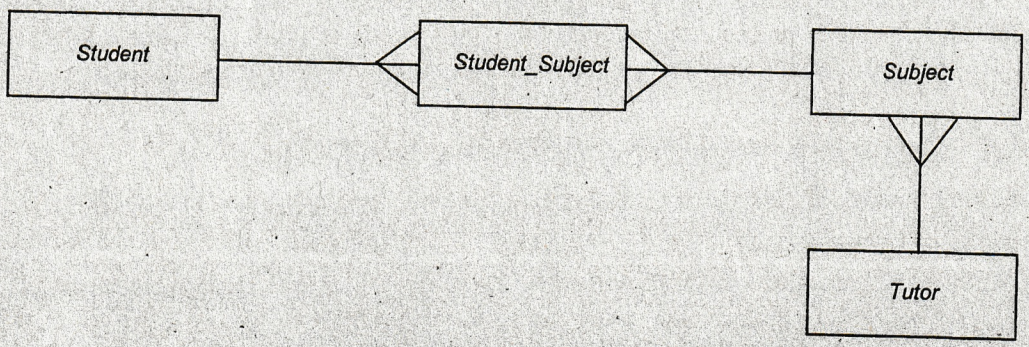

Figure 1.3: E-R diagram for the Student database application

The four 'normalised' tables that we need for this application are, therefore

STUDENT (StudentNumber, Surname, FirstName, DateOfBirth, DateEnrolled)

SUBJECT (SubjectCode, SubjectName, HrsPerWeek, TutorID)

STUDENT_SUBJECT (StudentNumber, SubjectCode)

TUTOR (TutorID, TutorName)

Many-to-many relationships cannot be directly implemented in a normalised database. An extra table always has to be introduced to link the two original entities. The database holding information about students and subjects needed the extra *link* table called STUDENT_SUBJECT. Notice carefully which way round the 'crowsfeet' point. It will always be the same as that shown below when you introduce a link table.

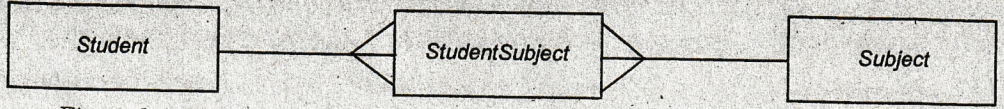

Figure 1.4: Using a 'link' table to join two entities in a many-to-many relationship

A sample database

Most of Part 1 will use as an example of Access techniques, a simple system to help a newsagent keep track of newspaper deliveries and customer accounts. Details will be held on customers and newspapers, and which newspapers the customers have delivered to them each day.

The objectives of the system are to be able to:

- Add, edit and delete details of customers, newspapers and deliveries;
- Print a 'round sheet' showing which newspapers are to be delivered to each customer on a given round each day;
- Print a summary report showing the total number of each newspaper required for each round;
- Calculate each customer's weekly bill and add it to their outstanding balance;
- Print each customer's weekly invoice;
- Update the customer's record when payment is received.

In Part 1, this case study is simply a vehicle for learning Access, so many other tasks will be tackled along the way and some parts of this system will not be implemented. However, you can see the complete system implemented in the sample project in Appendix A, and you can download the Access 97 database **newsproject.mdb** from the web site www.payne-gallway.co.uk

You can also download the database as it should be at the end of chapters 2-10 in files called **newsagt2.mdb**, **newsagt3.mdb**, etc.

Identifying the entities, attributes and relationships

The first step in the design is to identify the *entities* involved. What or who are we holding data about? The most obvious entities in this application are CUSTOMER and NEWSPAPER.

Next, how are these two entities related? Any customer can have many newspapers delivered, and the same newspaper (e.g. the Daily Mail) may be delivered to many different customers. Therefore, there is a many-to-many relationship between these two entities.

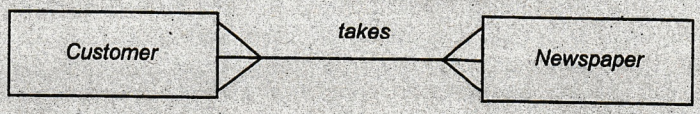

Figure 1.5: Relationship between customers and newspapers

As we saw above, a many-to-many relationship cannot be directly implemented and we will need a 'link' table to specify which newspapers each customer wants delivered. The third table could be named DELIVERY, and the new E-R diagram will be as follows:

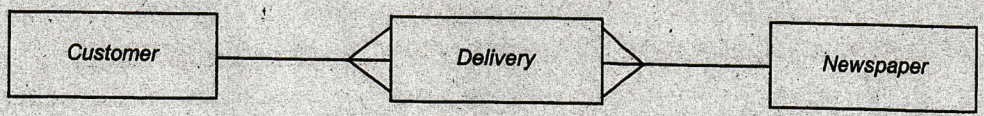

Figure 1.6: E-R diagram for the Newspaper application

If you were to perform a more detailed analysis, you might decide there were other entities which required tables of their own – for example Round, Newspaper Boy/Girl, Supplier etc. In this application, however, we will use just the three tables for the entities CUSTOMER, DELIVERY and NEWSPAPER.

Table design

The next stage in design is to decide what data we are going to hold in each table, what format the data should be in, what validations need to be performed on the data, and whether we can set any default values to minimise data entry.

Before you design the tables for your own project, you will need to be familiar with the various data types available in Access. The most common data types are:

Text	Up to 255 characters for holding a name, address, telephone number etc. (A telephone number is normally held as text because although it consists of numbers, it may contain spaces or leading zeros and is never used in calculations.)
Number	For numeric data which is to be used in calculations. Access allows different types of number field such as **byte** (for numbers between 0-255) **integer** (for whole numbers between -32,768 and 32,767), **long integer** (for larger whole numbers), **single** and **double** for numbers with decimal points, and more.
Autonumber	A unique sequential (incrementing by 1) or random number is automatically inserted when a record is added. Commonly used as a key field.
Currency	For holding currency amounts – use it to prevent rounding during calculations.
Boolean	For fields containing only Yes/No or True/False.
Date/Time	For holding dates and/or times.
Memo	For lengthy notes of up to 64,000 characters.

The tables will be defined as follows:

CUSTOMER Table name: **tblCustomer**

Attribute Name	Data Type	Description/Validation
CustomerID*	AutoNumber	Key field, automatically incremented for each new customer
Surname	Text (20)	
Initial	Text (1)	
Title	Text (4)	Lookup from list – Mr, Mrs, Miss, Ms, Dr, Rev
Address1	Text (20)	1st line of address
Address2	Text (20)	2nd line of address
Round	Integer	1-3
CurrentDue	Currency	This week's amount due
PastDue	Currency	Past amount due
HolsBegin	Date	Date holiday starts
HolsEnd	Date	Date holiday ends

NEWSPAPER Table name: **tblNewspaper**

Attribute Name	Data Type	Description/Validation
NewspaperID*	Text (4)	Key field
NewspaperName	Text (25)	
Price	Currency	
Morning/Evening	Text (1)	M or E

DELIVERY Table name: **tblDelivery**

Attribute Name	Data Type	Description/Validation
CustomerID*	Long Integer	Part of the key field – must exist on CUSTOMER table
NewspaperID*	Text (4)	Part of the key field – must exist on NEWSPAPER table
Monday	Yes/No	Y/N indicates whether paper delivered on Monday
Tuesday	Yes/No	As for Monday
Wednesday	Yes/No	"
Thursday	Yes/No	"
Friday	Yes/No	"
Saturday	Yes/No	"
Sunday	Yes/No	"

Figure 1.7: Designs for the 3 tables CUSTOMER, NEWSPAPER and DELIVERY

Elements of an Access database

The various elements that you'll be working with in Access are referred to as *objects*. These include:

Tables for holding information;
Queries for asking questions about your data or making changes to it;
Forms for editing and viewing information;
Reports for summarising and printing information;
Macros for performing tasks automatically;
Modules for customising your database using Visual Basic for Applications (VBA).

Naming conventions

There are various conventions for naming the objects that you use. You don't have to use a naming convention but it will certainly make your database easier to create and maintain, and will probably earn you extra marks in project work. Shown below are the Leszynski/Reddick naming conventions, which will be used in this book.

Level 1

Object	*Tag*	*Example*
Table	tbl	tblCustomer
Query	qry	qryClientName
Form	frm	frmCustomer
Report	rpt	rptSales
Macro	mcr	mcrUpdateList
Module	bas	basIsNotLoaded

Level 2

Object	*Tag*	*Example*
Table	tbl	tblCustomer
Table (lookup)	tlkp	tlkpRegion
Table (system)	zstbl	zstblUser
Query (select)	qry	qryClientName
Query (append)	qapp	qappNewPhone
Query (crosstab)	qxtb	qxtbYearSales
Query (delete)	qdel	qdelOldCases
Query (form filter)	qflt	qfltAlphaList
Query (lookup)	qlkp	qlkpSalary
Query (make table)	qmak	qmakSaleTo
Query (system)	zsqry	zsqryMacroName
Query (update)	qupd	qupdDiscount
Form	frm	frmCustomer
Form (dialogue)	fdlg	fdlgInputDate
Form (menu)	fmnu	fmnuMain
Form (message)	fmsg	fmsgCheckDate
Form (subform)	fsub	fsubInvoice
Report	rpt	rptTotals
Report (subreport)	rsub	rsubValues
Report (system)	zsrpt	zsrptMacroName
Macro	mcr	mcrUpdateList
Macro (for form)	m[formname]	m[formname]Customer
Macro (menu)	mmnu	mmnuStartForm
Macro (for report)	m[rptname]	m[rptname]Totals
Macro (system)	zsmcr	zsmcrLoadLookUp
Module	bas	basTimeScreen
Module (system)	zsbas	zsbasAPIcall

Figure 1.8: Leszynski/Reddick naming conventions

Designing the Input and Output

When you start your own project, you will be expected to design the whole system including the menu, input forms, queries, processing and reports before you start implementation. However, the purpose of Part 1 of this book is to teach you how to use Access and what its capabilities are, so the design of each part of the system will be explained as we come to implement it.

We're ready to load Access and create the tables!